Attacks on Implementations of Secure Systems

June 12, 2019

Chapter 1

Introduction

Once upon a time... This document shows how you can get ePub-like formatting in LaTeX with the `memoir` document class. You can't yet export directly to ePub from writeLaTeX, but you can download the source and run it through a format conversion tool, such as `htlatex` to get HTML, and then go from HTML to ePub with a tool like Sigil or Calibre. See http://tex.stackexchange.com/questions/16569 for more advice. And they lived happily ever after.

Contents

Chapter 2

Writing LaTeX

2.1 Basic Formatting

Comments. If you want to just add a comment to a file without it being printed, add a % (percentage) sign in front of it. In the template files, you will find a number of such comments as well as deactivated commands.

Bold formatting. You can make your text bold by surrounding it with the command `\textbf{}`.

Italics formatting. You can make your text italic by surrounding it with the command `\textit{}`.

Small caps. You can change your text into small capitals by surrounding it with the command `\textsc{}`.

Text em dashes. Em dashes are used to connect two related sentences. There is no space before or after the em dash. Within the template, use the command `\textemdash{}` instead of using the dash you copied over from your text file. This will also take care of issues relating to line breaks.

Paragraphs. Paragraphs are handled automatically by leaving an empty line between each paragraph. Adding more

than one empty line will not change anything—remember it is not a "what you see is what you get" editor.

Empty line. If you want to force an empty line (recommended only in special cases), you can use `~\\` (tilde followed by two backslashes).

New page. Pages are handled automatically by LaTeX. It tries to be smart in terms of positioning paragraphs and pictures. Sometimes it is necessary to add a page break, though (ideally, at the very end when polishing the final text). For that, simply add a `\newpage`.

Quotation marks. In the normal computer character set, there are more than one type of quotation marks. It is required to change all quotation marks into ``...'' (two back ticks at the beginning and two single ticks at the end) and refrain from using "..." (or "...") altogether. This is because Word's "..." uses special characters, and "..." do not mark the beginning and end of the quotation.

Horizontal line. For a horizontal line, simply write `\hline`.

Underlined text. It is generally not recommended to use underlined text.

URLs. For URLs you need a special monospaced font. Also, for URLs in e-books, you want to make them clickable. Both can be accomplished by putting the URL in the `\url{}` environment, for example `\url{https://www.lode.de}`.

Special characters. If you need special characters or mathematical formulas, there is a whole body of work on that subject. It is not in the scope of this book to provide you a comprehensive list.

2.2 Lists

Itemized list. To create a bullet point list (like the list in this section), use the following construct:

```
\begin{itemize}
    \item Your first item.
    \item Your second item.
    \item Your third item.
%   \item Your commented item.
\end{itemize}
```

The result will look like this:

- Your first item.
- Your second item.
- Your third item.

Numbered list. To create a numbered list, replace itemize with enumerate:

```
\begin{enumerate}
    \item Your first item.
    \item Your second item.
    \item Your third item.
\end{enumerate}
```

The result will look like this:

1. Your first item.
2. Your second item.
3. Your third item.

2.3 Verbatim text

Sometimes, you do want to simply use text in a verbatim way (including special characters and LaTeX commands). For

this, simply use the `\lstlisting` environment: `\begin{lstlisting}...\end{lstlisting}`. For example, I put the itemize and enumerate listings above into a `\lstlisting` block. If I did not, LaTeX would have displayed the list as a list, instead of displaying the code.

2.4 Chapters and Sections

LaTeX uses a hierarchy of chapters, sections, and subsections. There are also sub-subsections, but for the sake of the reader, it is best to not go that deep. If you come across a situation where it looks like you need it anyway, I recommend thinking over the structure of your book rather than using sub-subsections.

In terms of their use in the code, they are all similar:

- `\chapter{Title of the Chapter}\label{c1_chaptername:cha}`
- `\section{Title of the Section}\label{c1_sectionname:sec}`
- `\subsection{Title of the Subsection}\label{c1_subsectionnam:sec}`
- `\paragraph{Title of the Paragraph}\label{c1_paragraph:sec}`

When using these commands, obviously replace the title, but also the label. For the label, I recommend to have it start with c, followed with the current chapter number, an underscore, and the chapter, section, or subsection in one word and lowercase, followed by either ":cha" or ":sec" to specify what kind of label it is. These labels can then be used for references like we used previously for the images. For example, if you have defined a section `\section{Chapters and Sections}\label{c1_chaptersandsections:sec}`, you could

write "We will discuss chapters and sections in section \↩ ref{c1_chaptersandsections:sec} which results in the document in "We will discuss chapters and sections in section 2.4".

2.5 Tables

In LaTeX, tables are like images and put into the figure environment. As such, they have a caption, label, and a positioning like we discussed above with the images. Drawing a table requires a bit of coding:

```
\begin{table}[!ht]
    \centering
    \begin{tabular}{p{2.5cm}|p{3.5cm}|p↩
       {3.5cm}}
    \hline
    & \textbf{Word} & \textbf{\LaTeX{}}↩
        \\
    \hline

    Editor & ``what you see is what you↩
        get'' & source file is compiled↩
        \\
    \hline

    Compatibility & dependent on editor↩
        & independent of editor \\
    \hline

    Graphics & simple inbuilt editor & ↩
       powerful but complex editor \\
    \hline

    Typography & optimized for speed & ↩
       optimized for quality \\
    \hline

    Style & inbuilt style & separate ↩
       style document \\
```

```
21      \hline
22
23      Multi-platform & only via export & ↩
           possible with scripting \\
24      \hline
25
26      Refresh & some elements need, ↩
           manual refresh & everything is ↩
           refreshed with each compile \\
27      \hline
28
29      Formulas & basic support needs ↩
           external tools & complete ↩
           support \\
30      \hline
31
32      \end{tabular}
33      \caption{Comparison of Word and \↩
           LaTeX{}} \label{c1_↩
           comparisonwordlatex:tab}
34  \end{table}
```

This table from the beginning of the book has the familiar figure, label, caption, and centering commands. The actual table is configured with the `\tabular{}` environment. Following the tabular command, you configure the columns in curly braces. Each column is separated with a vertical line and the `p{...}` entry specifies the width of the column. With `{p↩{2.5cm}|p{3.5cm}|p{3.5cm}}`, you would have three columns with 2.5cm width for the first column and 3.5cm width for the two others. Alternatively, you can use `c` instead of `p` and leave out the curly braces with the width. Then, LaTeX simply calculates the required widths automatically. Then, for each line of the table, simply write: `content of the first cell & ↩ content of the second cell & content of the third cell↩ \\\hline`.

	Word	**LaTeX**
Editor	"what you see is what you get"	source file is compiled
Compatibility	dependent on editor	independent of editor
Graphics	simple inbuilt editor	powerful but complex editor
Typography	optimized for speed	optimized for quality
Style	inbuilt style	separate style document
Multi-platform	only via export	possible with scripting
Refresh	some elements need, manual refresh	everything is refreshed with each compile
Formulas	basic support needs external tools	complete support

Table 2.1: Comparison of Word and LaTeX

2.6 Footnotes

Finally, for footnotes, there is the command `\footnote{}`. You can place it anywhere you like, LaTeX will then automatically add the number of the footnote at that place, and put the footnote text into the footer area. It looks like this.[1] The challenge here relates to grammar: footnotes start with capital letters, parentheses with lower case, and the footnote comes after the period, the parentheses have to start before the period.

[1] This is a footnote.

Chapter 3

Inserting Images

3.1 Images

As in Word, in LaTeX, images are separate from the text. Images are usually packaged together with a caption and a label to reference it from the text. These three entities are packaged together into a figure. The figure itself configures the size of the image as well as where it should be put. Let us look at a code sample:

```
1  \begin{figure}[H]
2      \centering
3      \includegraphics{images/ebookLatex_↩
          Cover.jpg}
4      \caption{The cover of this book.} \↩
          label{c1_cover:fig}
5  \end{figure}
```

Let us go through this line by line. At the core is the image, included with `\includegraphics{path to file}`. It inserts the image specified by the "path to file." With the `\adjustbox{}` command, we can adjust the image size according to the page width (`\columnwidth`) and page height (`\textheight`).

Below there is the caption and the label. LaTeX automatically numbers each figure, so in the text, we can later refer to

it with `\ref{c1_cover:fig}` which prints out the number of the figure. Finally, all these commands are centered with the `\centering` command and surrounded with the figure environment. The `[ht]!` instructs LaTeX to try to place the image exactly where it is in the LaTeX code.

Figure 3.1: The cover of this book.

In Figure 3.1, you can see the result of the command. Instead of graphics, you can also include other TEX files that contain graphics (or commands to draw graphics, see chapter 3.2).

3.2 TikZ Graphics

For graphics, you can use the inbuilt TikZ graphics generator. Due to its flexibility, I even recommend images you already

have for a number of reasons:

- TikZ graphics can very easily changed (especially for for example translations or making corrections).
- TikZ graphics are small and flexible. They can be easily scaled to any size and are directly integrated into your project (no time-consuming editing in an external graphics program necessary).
- TikZ graphics look better. As vector graphics are sent directly to the printer, we need not to worry about readability.

If you want to create a TikZ graphic, simply create a new TEX file in the *tex-images* folder and include it with `\input` (replacing `\includegraphics{}`) where you want to.

Then, do a "recompile from scratch" by clicking on the top right corner of the preview window (showing Warning or Error) to regenerate the TikZ file. If "up-to-date and saved" is shown, delete the *tikz-cache* directory and recreate it.

For the format of the file itself, it is a series of commands surrounded by the `\begin{tikzpicture}...\end{tikzpicture}` environment. Discussing all the commands is beyond the scope of this book, so I recommend three options:

- Check out the PGF manual at https://www.ctan.org/pkg/pgf. It is more than 1100 pages full with documentation of each command and corresponding examples.
- Check out the few example TikZ pictures from my two books [**?**] and [**?**] in the *tex-images* directory.

Chapter 4

Replace with Third Chapter Name

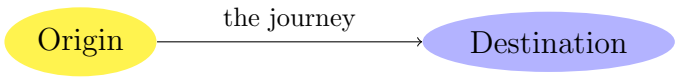

Figure 4.1: TikZ drawings will be output as SVG, which should be rendered by most modern browsers.

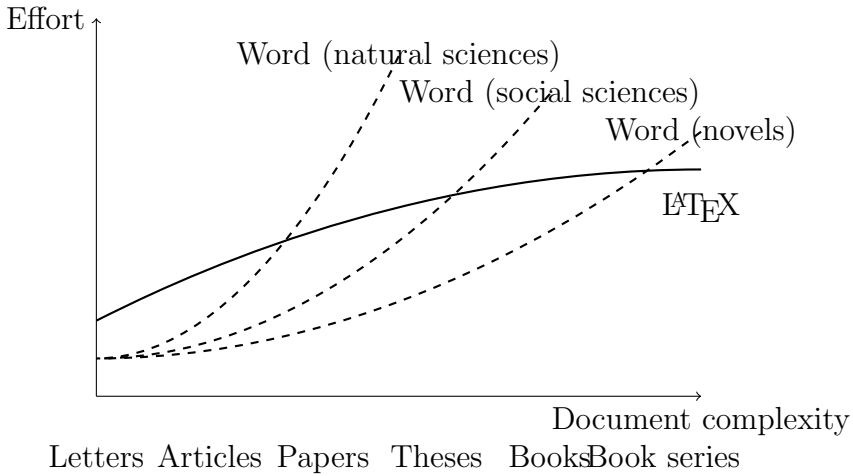

Figure 4.2: Comparing complexity of *Word* and LaTeX depending on the application.

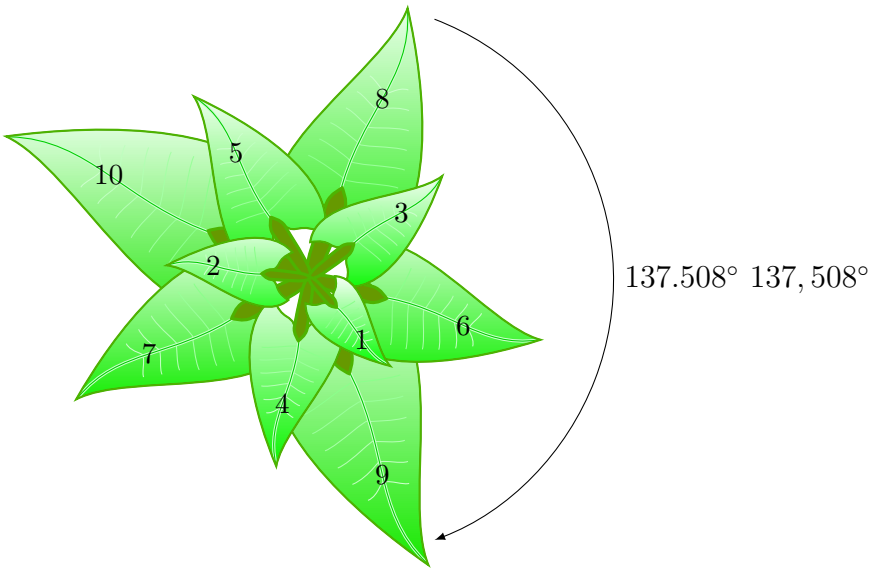

Figure 4.3: Example of a drawing made in TikZ.

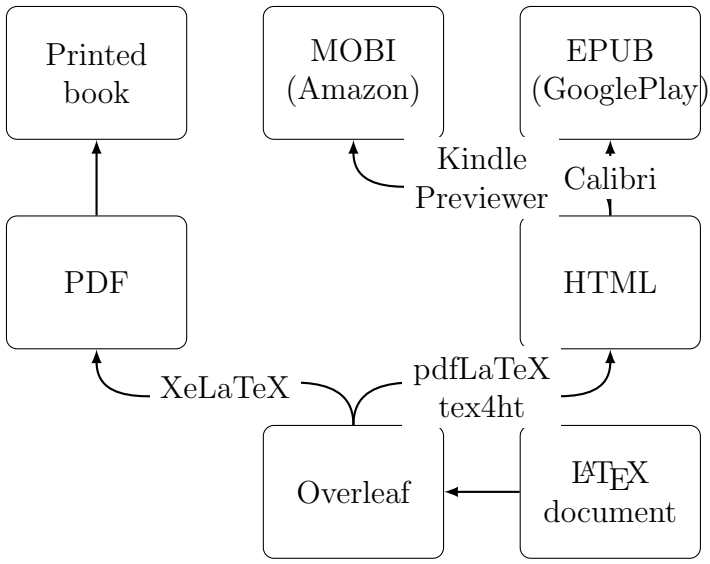

Figure 4.4: Example 2 of a drawing made in TikZ.

Chapter 5

Replace with Chapter 4 Name

Chapter 6

Replace with Chapter 5 Name

Chapter 7

Replace with Chapter 6 Name

Chapter 8

Replace with Chapter 7 Name

Chapter 9

Replace with Chapter 8 Name

Chapter 10

Fault Attacks

372-2-5421

Attacks on Implementations of Secure Systems

Scribe notes – Fault Attacks

Dorel Yaffe, Dan Arad, Iliya Fayans

{yaffed,arda,fayansi}@post.bgu.ac.il

2019-05-27

Topics

1. Introduction to Fault Attacks
2. Fault Attack on RSA-CRT
3. DRAM and Rowhammer
4. Flip-Feng-Shui: Rowhammer attack on RSA

Introduction to Fault Attacks

Up until now in the course we learned about *passive* attacks – i.e. attacks which measure *leakage* such as timing information and power traces. The advantage of these attacks is that they allow an attacker to acquire information in the process of an ongoing computation e.g. an AES key *before* it was fully mixed with the input – this fact can help the attacker extract secrets.

In fault attacks we will become *active* in the sense that we will give the device-under-test (DUT) additional inputs such as heat or radiation.

One problem with Fault attacks: they use the strongest attack model, meaning – we assume most power on the attacker's part.

10.1 Active Attacks

Definition:

> *"A fault attack is an active attack that allows extraction of secret information from cryptographic devices by breaking those devices."*

In fault attacks we are being *active* – we give additional inputs beside the main input such as:

1. Fuzzing (garbage or bad input)
2. Radiation
3. Heat
4. Vibration
5. Power spikes etc.

This way, we receive other (usually erronous) outputs which might give us additional information about the computation and/or the secret. This process is described in figure 10.1.

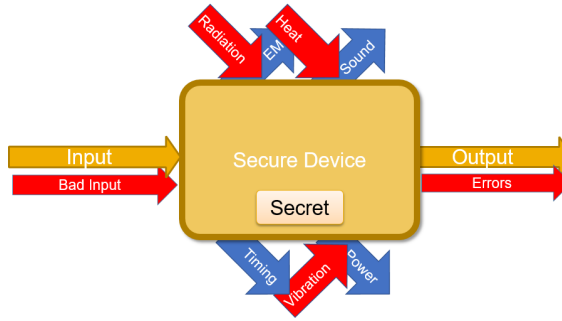

Figure 10.1: A schematic diagram of fault attacks and leakage types.

Like with passive attacks, some of those outputs can be acquired at different stages of the computation process.

Many fault attacks are inspired by studies in the field of *reliability*: a study in reliability will research a device's physical boundaries e.g. the maximum or minimum temperature under which it performs reliably. Other examples of reliability studies are aimed at improving device performance under extreme conditions such as:

- Space and X-Rays
- Dense CPU layouts
- Datacenter recovery (ECC-RAM)

A security researcher implementing fault attacks will, on the other hand, purposefully subject the DUT to extreme conditions in order to inject errors in the device's functionality to achieve their goal. In that sense, a reliability study of a given device can lay the ground-work for the fault attacks to come.

> *"In the reliability community things happen by mistake. In the security community – things happen on purpose."*

Breaking a device-under-test

How can *breaking* a device help an attacker?

1. BORE – *"Break Once, Run Everywhere"*: Some device families share a single secret among all instances.
2. Repairable Devices: Temporary breakage is fine. Sometimes restarting the device is enough to "repair" the damage.
3. Partial breakage: Sometimes it's convenient to break *part* of a device, for example – destroy the subsystem responsible for DRM verification.

10.2 Fault Attack Taxonomy

The tree ways we can examine a Fault Attack in order to understand it are:

1. Method - *"How to inject?"*
2. Properties - *"What fault to create?"*
3. Target - *"Which part to break?"*

Fault Methods

Power Supply Attacks

What happens if we underpower a device? As we have previously seen, power in electronic devices is used to drive the CMOS transistors. If the device is slightly underpowered it might fail to switch some of the transistors and thus produce false calculations, and with even less power it might struggle

with getting into operational state (boot loop) or even transition to an entirely faulty state. Another attack method involving the power supply is injecting power spikes (to a similar effect).

Some parts of a device are typically more sensitive to the power supply than others, and thus under- or over-powering the device will de-stabilize it and inject faults.

The obvious scenario for such an attack is when the DUT belongs to or is being controlled by the attacker – for example if they're examining their own set-top box etc. In that case, the attacker can supply the device with as much/little power as they wish.

Another example of such attack scenarios is in the field of RFID readers – the device powering an RFID chip is the reader, so a *malicious* RFID reader can over/under-power the chip to achieve various faulty results.

Clock/Timing Attacks

The clock is typically a bus shared by many of the system's components which synchronizes the propagation of calculations through the system – i.e. at the beginning all inputs are ready, and when there is a rising edge on the clock bus they start propagating throught the various computational components. When all computations are finished they all wait for the next rising edge on the clock bus in order to proceed to the next stage. In a clock glitching attack the attacker would inject a rising edge on the clock bus at an arbitrary time. This way only *some* of the computations will have finished by that time while others are still being processed, and thus the device will be in a faulty (unstable) state.

A notable example is the attack on Mifare Classic RFID chips we talked about in the beginning of the course [1] – the RNG in the chip is only dependent on the time between powering up the RFID tag and challenging it. An RFID reader operated by the attacker can control both parameters, thus making the generated challenges deterministic.

Temperature attacks

This attack method relies on the physical property of electrons (current). Electrons "jump", and the higher the temperature – they will jump more frequently and to longer distances. If a device gets *too hot* – enough electrons can "jump" e.g. over the insulation layer in a transistor to flip it from logical 1 to 0. This results in a fault.

Because of the ubiquity of devices failures due to temperature, nowadays temperature sensors are integrated into most devices, so when it overheats – the device will shut down.

An attacker can bypass the temperature sensor by disconnecting it. Another method would be to quickly alternate the temperature of the device from extremely high to extremely low, so that *on average* the temperature is reasonable, but it will still experience faults during the extreme phases of the cycles.

In an interesting research paper [2] a type-confusion attack on the Java virtual-machine was demonstrated: at first, the entire memory was filled with small arrays (say of size one). The Java VM is type-safe, so it is normally impossible to access one of the memory regions using a pointer to a different region. To inject a type-confusion fault the researchers used a 50W lightbulb to heat the memory chip of the device enough to flip some of the bits (for illustration see Fig. 10.2). As a result, a small portion of the data-structures describing the arrays in memory now held wrong values (e.g. changed their value from $size = 1$ to $size = 20$). At this point, some affected data-structure *contains* a header of a different data-structure, to which the attackers now have read and write access. Changing the header of the second data structure to an arbitrary value gave the attackers access to the entirety of the device's memory.

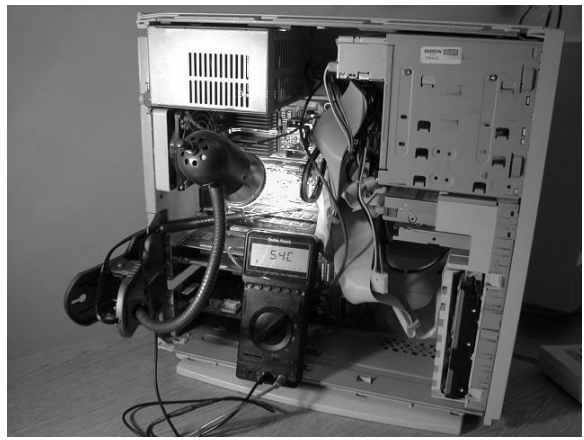

Figure 10.2: A lightbulb flipping memory bits filled with safe Java structures.

Optical, Electromagnetic

When a laser hits a transistor it changes the energy level of the silicon inside, and sometimes it can change the transistor's state. Notably different wavelengths are absorbed by different materials, so in a typical silicon chip different lasers will hit different layers of the device etc. Magnetic/Electromagnetic radiation and pulses have similar effects.

The underlying principal of those attacks is that the attacker forcefully injects charge (energy) into the device. Once it's stored inside it will have to dissipate one way or another, so as a result it injects a random faulty state into the device.

Reading from RAM

All of the attacks described above require very high engagement with the DUT – in order for the attacker to control the power/clock sources, for example, they sometimes would need to drill, cut or othewise tamper with the device. Shining a laser on a device requires at the very least having it at a visible distance.

The final attack method involves only *reading* from memory,

and thus is very practical and requires very little physical engagement. This attack method is called *Rowhammer* and is discussed later in the lecture.

Fault Properties

In this section we discuss:

1. How controllable is the fault's location/size? Precise? Loose? None?
2. How controllable is the fault timing?
3. What's the fault's duration? Transient? Permanent? Destructive?

Destructive fault attacks on cryptographic devices

What can be done with fault attacks to symmetric ciphers?

Easy example: Imagine that we have a pile of cipher devices with a 64bit key length, which work the following way: we can give the device a key and it tells us whether it's the right key.

What if we have a DESTRUCTIVE fault attack that anulls the top 32bits of a device's key? We can brute force the key in $2 \cdot 2^{31}$ instead of 2^{63} (on average):

First we inject the fault into one of the devices and brute-force the lower 32 bits of the key ($O(2^{31})$), then we pick another device from the pile and brute-force only the higher 32 bits (another $O(2^{31})$).

A less trivial example: We have a public-key device which we can ZAP and one bit of the key flips to zero.

We can save all of the plainexts-cipher pairs until we reach the one matching an all zero key – which we can pre-calculate. This gives us the Hamming weight of the key. Now we go back one plaintext-cipher pair – we know that pair's key's

Hamming weight to be exactly one – meaning we need to brute-force only N keys (N is the key bit-length). Now we have the position of a single bit of the key.

If we itreate all the way backwards to the original plaintext-cipher pair, we can acquire the key in $O(N^2)$ time!

Fault Target

What could be targeted by a fault attack?

1. Input parameters (fuzzing, clock glitching)
2. Storage (volatile/non-volatile)
    a) HDD - Destructive (persists after reset)
    b) RAM - Permanent
    c) Cache - Transient
3. Data processing: inject a fault mid-computation and the device gives a differnt answer.
4. Instruction Processing/Control Flow: inject a fault in the IP register and change the instruction flow.
    a) ARM32 instructions are very densely packed, thus there is a very high probability of hitting a valid instruction after flipping a single bit. `jnz` and `jmp` are only one bit apart.
    b) Change for loop condition to dump RAM contents including source code.

Two examples of Fault Attacks targeting Control Flow:

1. The CHDK hacking community, used to dump the firmwares of Canon cameras via blinking one of their LEDs [3, 4].
2. "The Unlooper": Back in the 90's pay-tv devices started cryptographically signing the content, and if the cryptographic checksum did not check out – the device

would enter an endless loop. The hacking community invented "unloopers" – a gadget that would inject a power spike and fault the IP register, so that the pay-tv device would jump to some other section of the code, from which point it would function normally.

10.3 Fault attack on RSA-CRT

RSA decryption

$$N = p \cdot q$$

$$M = C^d (mod\ n) = M^{ed} (mod\ n) = M^1 (mod\ n)$$

RSA decryption is hard!

Let's speed it up using CRT (the Chinese Remainder Theorem): Multiplication operations are $O(|n|^2)$. If we can do operations $(mod\ p)$ and then $(mod\ q)$ instead of $(mod\ n)$, we will reduce computation time by half.

Explanation: The bit-lengths of p and q are each half that of n

$$|p| = |q| = \frac{1}{2}|n|$$

The computational complexity of multiplying by a number of length x is (roughly) $O(x^2)$. Thus:

$$O(|p|^2) = O(|q|^2) = \frac{1}{4}O(|n|^2) \Rightarrow (O(|p|^2) + O(|q|^2)) = \frac{1}{2}O(|n|^2)$$

So if we could multiply by p and q instead of by n, we would cut *each* multiplication operation's time complexity in half!

Chinese Remainder Theorem

The idea is that if we know both $x(mod\ p)$ and $x(mod\ q)$ then we can easily calculate $x(mod\ n)$.

So, given a message M calculate M_p and M_q:

$$M_p = C^d (mod\ n)(mod\ p) = C^d (mod\ p)$$

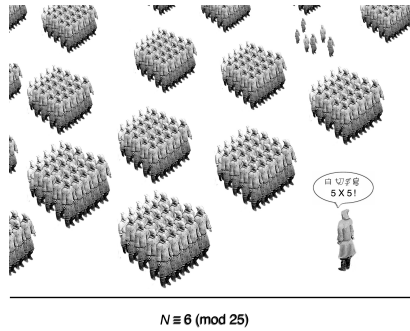

Figure 10.3: Chinese Remainder Theorem (Source: https://russinoff.com/papers/crt.html)

$$M_q = C^d(mod\ n)(mod\ q) = C^d(mod\ q)$$

To combine the values, we do:

$$M^* = CRT(M_p, M_q) = M_p \cdot q \cdot (q^{-1} mod\ p) + M_q \cdot p \cdot (p^{-1} mod\ q)$$

It is easily provable that $M^*(mod\ p) = M_q$ and $M^*(mod\ q) = M_p$, so by the Chinese Remainder Theorem, this value *must* be equal to M.

The Boneh, DeMillo & Lipton Fault Attack on RSA-CRT [5]

The attacker has a decryption box (known plaintext scenario) with public key n and would like to recover d (the private key). Additionally, the attacker knows that the decryption box is decrypting using CRT. Finally, let us assume that the attacker can inject a fault (any fault) in the decryption process.

The attacker first gets $M = M_p * q * (q^{-1} mod\ p) + M_q * p * (p^{-1} mod\ q)$ through the regular decryption process.

Then, the attacker primes the device to re-calculate the message from the same cipher, this time injecting a *fault* during the calculation of M_p, resulting in the device erronously pro-

ducing M'_p instead:

$$M'_p \neq C^d (mod\ p)$$

The device will then proceed to combine M'_p with the correct result of M_q, resulting in:

$$M' = M'_p \cdot q \cdot (q^{-1} mod\ p) + M_q \cdot p \cdot (p^{-1} mod\ q)$$

Now the attacker can calculate the value of $M - M'$:

$$(M_p \cdot q \cdot (q^{-1} mod\, p) + M_q \cdot p \cdot (p^{-1} mod\, q)) - (M'_p \cdot q \cdot (q^{-1} mod\, p) + M_q \cdot p \cdot (p^{-1} mod$$

$$(M_p - M'_p) \cdot q \cdot (q^{-1} mod\ p))$$

Finally, calculating the *gcd* of n and $M - M'$ yields:

$$\gcd(n, M - M') = \gcd(p \cdot q, (M_p - M'_p) \cdot q \cdot (q^{-1} mod\ p)) = q$$

Why does this work? The greatest common divisor of n and anything can be only p, q, n or 1. On the other hand, M_p and M'_p can never be multiples of p, otherwise both would equal 0. So, by that reasoning, $\gcd(p \cdot q, (M_p - M'_p) \cdot q \cdot (q^{-1} mod\ p))$ must equal q, and thus we have cracked the cipher using a single fault attack.

A later paper co-written by Arjen Lenstra [6] further improved upon this attack to not require knowledge of M.

BML in practice: A paper [7] showed how ZAPPING a device with an electric spark from a lighter during computation can achieve the described effect.

10.4 Rowhammer

In the final section, we will describe the Rowhammer attack.

Rowhammer attack taxonomy

- Target: DRAM on modern computers
- Properties: Permanent, controlled location
- Method: Memory accesses

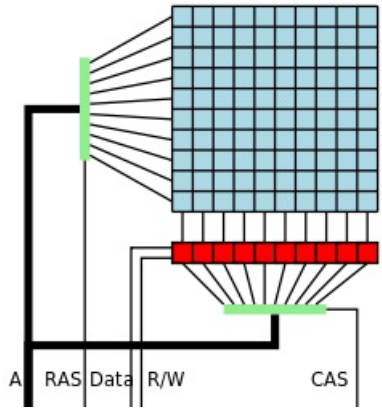

Figure 10.4: High Level Illustration of DRAM Organization (Source: Wikipedia:Row Hammer)

Double-sided Rowhammer

DRAM is the most common type of volatile memory. It is slow, dense and cheap relatively to SRAM. Every bit of RAM is stored in a single capacitor.

The attack [8] utilizes the physical structure of RAM chips in order to induce faults: Due to parasitic leakage in DRAM capacitors, if enough consecutive reads are performed on the immediately adjacent rows eventually a bit will flip in the target row.

The challenge of CPU caching

The CPU cache prevents the same memory address to be read consecutively from main memory, for performance rea-

sons. To circumvent this limitation, several techniques can be employed:

1. Intel CPUs provide non-temporal read/write opcodes – special instructions that read from memory and don't get cached.
2. The special `clflush` instruction can be used to explicitly flush the cache after each read operation (priviledged operation).
3. Finally, cache-population algorithms had been extensively studied and reverse-engineered, so it is possible to arrange for *arbitrary* cache misses.

Countermeasures

Refresh-rate increase: In order to overcome parasitic leakage, DRAM chips already have a mechanism in place to read and then re-write the values stored in each row periodically. One method of overcoming Rowhammer could be to significantly increase the refresh-rate of the chip. This obviously results in both performance degradation and increased power consumption.

ECC-RAM: High-end DRAM chips (typically meant for datacenter environments) contain error-coorection coding (ECC) logic which can typically *correct* one erronous bit and *detect* two (at which point it will crash the program/system). Those chips are immune to the basic form of Rowhammer described above, but as discussed later, are not bullet-proof.

Rowhammer variations

Flip Feng-Shui

Page de-duplication: On modern systems, typically much memory is shared among many processes running on the system. This is even more true of virtualized environments

where the guest and the host, for example, could run the same OS. A common optimization is for the system to detect and de-duplicate pages containing the same data, thus freeing up memory.

Rowhammer + page de-duplication: In a paper [9] researchers from VUA demostrated how they can utilize page-deduplication in a virtualized environment to weaken cryptographic keys, resulting in unauthorized access via OpenSSH, and breach of trust via forging GPG signatures. The attack relies on the fact that the the attacker can *read* a de-duplicated page as much as they want. The attacker has to wait (or arrange) for a page containing sensitive information to be de-duped, then hammer on it until a bit in the key flips, making it much easier to factor.

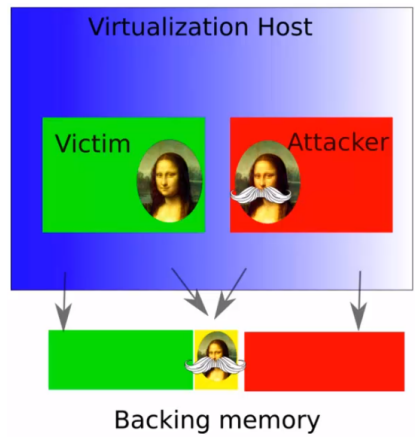

Figure 10.5: The attacker maps the same page as the victim, then utilizes rowhammer to change the victim's memory without causing page duplication.

ECCPloit

In another paper [10] researchers from VUA showed how they can use Rowhammer to quicly flip *enough* (typically three) bits on an ECC-RAM chip that error correction will not be

able to detect it, thus defeating the ECC mitigation. The attack relies on the fact that error-correction takes time to compute, and this gives the attacker a window of opportunity.

Bibliography

[1] K. Nohl et al. Reverse-engineering a cryptographic rfid tag. In *17th USENIX Security Symposium*, volume 17. USENIX, 2008. Online; https://www.usenix.org/legacy/events/sec08/tech/full_papers/nohl/nohl.pdf.

[2] A. Govindavajhala, S. & Appel. Using memoryerrors to attack a virtual machine. Princeton University. Online; https://www.cs.princeton.edu/~appel/papers/memerr.pdf.

[3] CHDK Wiki. Obtaining a firmware dump. CHDK Wiki. Online; https://chdk.fandom.com/wiki/Obtaining_a_firmware_dump#Q._How_can_I_get_a_firmware_dump.3F.

[4] HackingHood. Canon sd300 camera analysis. HackingHood. Online; https://sites.google.com/site/hackinghood2/home.

[5] D. Boneh et al. On the importance of eliminating errors in cryptographic computations. Dept. of Computer Science, Stanford University. Online; https://link.springer.com/content/pdf/10.1007/s001450010016.pdf.

[6] M. Joye et al. Chinese remaindering based cryptosystems in the presence of faults. UCL Crypto Group, Dep. de Mathematique, Universite de Louvain. Online; http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.5491&rep=rep1&type=pdf.

[7] Schmidt J.M. & Hutter M. Optical and em fault-attacks on crt-based rsa: Concrete results. Institute for Applied Information Processing and Communciations, Graz University of Technology. Online; https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=32877.

[8] Y. Kim et al. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. Intel Labs, Carnegie Mellon University. Online; https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf.

[9] K. Razavi et al. Flip feng shui: Hammering a needle in the software stack. Vrije Universiteit Amsterdam. Online; https://www.cs.vu.nl/~herbertb/download/papers/flip-feng-shui_bheu16.pdf.

[10] L. Cojocar et al. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. Vrije Universiteit Amsterdam. Online; https://cs.vu.nl/~lcr220/ecc/ecc-rh-paper-sp2019-cr.pdf.